Assignment 03 - Edges and filters

This assignment may be viewed in the original markdown format or as a pdf (assignment03.pdf) which renders the equations in a readable form. If you're curious, the pdf was created using a tool called pandoc which "compiles" from markdown to LaTeX then from LaTeX to pdf.

Problem 1: Separable filters (15 points)

1. Let $\mathbf{h} = \mathbf{u}\mathbf{v}^\top$ be a separable 2D filter constructed as the outer product of a column vector $\mathbf{u}$ and row vector $\mathbf{v}^\top$. In other words, $\mathbf{h}_{k,l} = \mathbf{u}_k \mathbf{v}_l$. Prove that if $\mathbf{u}$ and $\mathbf{v}$ are each normalized (so $\sum_{k=1} \mathbf{u}_k = 1$ and $\sum_{l=1} \mathbf{v}_l = 1$), then the 2D filter $\mathbf{h}$ is also normalized (so $\sum_{k,l} \mathbf{h}_{k,l} = 1$).

   You can either write the proof in this document using markdown and LaTeX formatting (putting equations inside dollar signs as we did above), or write your proof as legibly as possible on paper and upload a picture.

   Proof/name of file with picture of proof: ________________

2. Correlation can be done with the OpenCV function cv.filter2D. Convolution can be done by first flipping the kernel (e.g. conv_kernel = cv.flip(corr_kernel, −1)) then using filter2D. However, in this problem you're going to implement your own version of correlation to see how it works under the hood and to better appreciate separability and the power of C loops over Python loops.

   Write the body of the my_correlation function in correlation.py and fill out any other ... parts of the file. The my_correlation function should make use of the numpy module but may not call any cv functions. Note that you will have to implement your own padding in order to match cv.BORDER_REPLICATE. (Hint: to replicate, you don't actually need to create a new padded image, you just need to re-use pixels from the border any time the index would take you outside the [0:h, 0:w] range). Be careful not to modify the original image in this function!

   Run it with a few different filter sizes and note the output for how long it takes to run using the 2d filter or two passes with the separate horizontal and vertical filters. Sanity-check that the results are the same whether you use cv.filter2D or your function, and whether you use the 2d kernel or the two separable passes. (It's ok if they are not *exactly* the same due to floating point rounding errors and the like, but they should be very close.) Pay close attention to values near the border to see if you correctly reproduced the behavior of cv.filter2D with the borderType=cv.BORDER_REPLICATE flag set.

   See how fast you can make your correlation function.

   Use matplotlib to make a plot of the runtimes for each of the different methods. Plot runtime in seconds on the y axis and filter size on the x axis, using a log scale for both. Save the plot as correlation_runtime.png and upload it as part of your submission. This code will look something like the following:

```python
import matplotlib.pyplot as plt

filter_sizes = [3, 5, 7, 9, 11, ...] # whatever values you used for filter sizes
time_elapsed_cv_separable = [0.2, 0.4, 0.5, 0.7, ...] # whatever values you got from the tim
time_elapsed_cv_2d = [...] # whatever values you got from the timing experiment
time_elapsed_mine_separable = [...] # whatever values you got from the timing experiment
time_elapsed_mine_2d = [...] # whatever values you got from the timing experiment

fig = plt.figure(figsize=(6,4))  # The figsize argument is (width, height) in inches
plt.plot(filter_sizes, time_elapsed_cv_separable, label="CV␣(separable)")
plt.plot(filter_sizes, time_elapsed_cv_2d, label="CV␣(2d)")
plt.plot(filter_sizes, time_elapsed_mine_separable, label="Mine␣(separable)")
plt.plot(filter_sizes, time_elapsed_mine_2d, label="Mine␣(2d)")
plt.xscale("log")
plt.yscale("log")
plt.xlabel("Filter␣size")
plt.ylabel("Runtime␣(s)")
plt.legend()
fig.saveas("correlation_runtime.png")
```

   If you want to get fancier, use multiple runs of each filter size, get the mean and standard deviation of each, and plot using plt.errorbar rather than plt.plot.

Questions: Do you see evidence of linear vs quadratic runtime? About how much faster is it to use the cv functions (which are calling compiled C code under the hood) versus calling pure python + numpy code?

Answers: ________________

Problem 2: Help Mario find the coins (15 points)

This problem is designed to give you some practice with template matching. We're going to help Mario find all the coins on the screen. The file mario.jpg contains a screenshot from a Mario game. The file coin.png contains a zoomed-in shot of one of the coins.

Modify the mario.py file to do the following:

1. Load the mario.jpg (scene) and coin.png (template) images. Note that this part is provided for you to handle splitting out the BGR and alpha channels of the mask.
2. Use the cv.matchTemplate function to correlate the scene and template. Use cv.TM_SQDIFF mode in order to use the mask as well (other template matching (TM) modes don't support masks).
3. Normalize the result so that the best-matching pixels have a value of 1.0 and the worst-matching pixels have a value of 0.0. (Note that unlike correlation, the SQDIFF function returns a *smaller* value for better matches!)
4. Look at the histogram of values you get out of step 3. Choose a sensible threshold value to use to decide whether a pixel is a match or not. (Hint: you can use the matplotlib library and plt.hist(a.ravel()) to plot a histogram of values in the 2d matrix a)

   What threshold value did you choose? ____________________
5. Set all values below the threshold to zero.
6. Implement the body of the non_maximal_suppression function inside of helpers.py
7. Call the non_maximal_suppression function on the thresholded mask to get points that are both above the threshold and are local maxima.
8. You should now have a "match image" that has just a few pixels with nonzero values. Use the np.where function or the cv.findNonZero function to get the (x, y) coordinates of these pixels.
9. Use the cv.putText function to overlay the number of coins found on the image in red text in the top left corner.
10. Use the cv.rectangle function to draw a red rectangle around each coin in the original image. Save the result as mario−matches.jpg, including both the coin count and the rectangles.
11. Make sure that all of the above runs inside of a if ___name___ == "___main___" block such that, if we were to replace mario.jpg and coin.png with new images, we could run python mario.py and it would generate the correct output

Now that you've helped mario out, let's think about generalizing and robustness. You don't need to implement any of the following, just respond to the questions below.

- is your coin detector robust to different screen resolutions? What if the mario.jpg image was twice as big or twice as small?
- is it robust to different scene brightness levels? How dark or bright can the image become before you stop detecting the coins?
- is it robust to different coin colors? What if the coins were blue instead of yellow?
- is it robust to different coin shapes? What if the coins were square instead of round?
- is it robust to different orientations? What if the coins were rotated by 45 degrees?

Discussion of robustness and invariance of template matching: ________________

Problem 3: Edge detection (15 points)

There are three files for this problem: edges01.jpg, edges02.jpg, and edges03.jpg. The first two are famous impressionist paintings and the third is an image of a brick wall. Your task is to tweak the parameters of the given Sobel edge detectors to get the best possible edge detection for each image based on your own subjective opinion. By "best possible" I mean that the edges should be clear and should convey the main semantic content of the image (e.g. boundaries between objects, clear shadows, etc). Details like the texture of the brush strokes or the rough surface of individual bricks should not be highlighted as edges.

Fill out the missing code in the file edges.py. The file runs as given, but the results are not very good. You should think about what operations you would want to add (e.g. downsampling, blur, thresholding, etc) and where in the pipeline you would want to add them. If these operations have parameters that govern their behavior (e.g. amount of blur, exact value of the threshold), add them as command line arguments to the script. For instance, if you add a blur operation, the script should be runnable like

python edges.py my_image.jpg –blur=5

Once you've settled on an edge detector you're happy with, do the following:

1. Below, briefly explain what modifications you made. What operations did you add? What parameters did you use? What was your general approach? You don't need to write a lot, just a few sentences to explain your thinking.

   Answer: ______________

2. What parameter settings did you use to get the best results for each image? (You can just copy and paste the command line you used to run the script for each image)

   Answer (edges01.jpg): ______________

   Answer (edges02.jpg): ______________

   Answer (edges03.jpg): ______________

3. Were you able to find one set of parameters that worked for all three images? If not, why not? What properties of the images themselves made it difficult to find a single set of parameters that worked for all three?

4. Run the script on all 3 of the above files and make sure that the output images are saved as edges01−out.jpg, edges02−out.jpg, and edges03−out.jpg respectively. Upload these images as part of your submission. We should be able to run the line of code that you put in part 2 above and get the same results as you did.

Finally, everything so far has been grayscale. Discuss how you would modify the code to detect edges in color images. What would you need to change or add? What would be your general approach? You don't need to implement this, just discuss some ideas.

Answer: ______________

Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did here.

Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?
- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Submitting

Your submission should consist of a single zip file named like firstname_lastname_assignment3.zip. The zip file should contain all the contents of this directory, including this file (which you should have written some answers into above) and any images or other outputs generated by your code.